On using Cholesky-based factorizations and regularization for solving rank-deficient sparse linear least-squares problems

SIAM J. SCI. COMPUT.
Vol. 39, No. 4, pp. C319–C339

ON USING CHOLESKY-BASED FACTORIZATIONS AND REGULARIZATION FOR SOLVING RANK-DEFICIENT SPARSE LINEAR LEAST-SQUARES PROBLEMS*

JENNIFER SCOTT†

Abstract. By examining the performance of modern parallel sparse direct solvers and exploiting our knowledge of the algorithms behind them, we perform numerical experiments to study how they can be used to efficiently solve rank-deficient sparse linear least-squares problems arising from practical applications. The Cholesky factorization of the normal equations breaks down when the least-squares problem is rank-deficient, while applying a symmetric indefinite solver to the augmented system can give an unacceptable level of fill in the factors. To try to resolve these difficulties, we consider a regularization procedure that modifies the diagonal of the unregularized matrix. This leads to matrices that are easier to factorize. We consider both the regularized normal equations and the regularized augmented system. We employ the computed factors of the regularized systems as preconditioners with an iterative solver to obtain the solution of the original (unregularized) problem. Furthermore, we look at using limited-memory incomplete Cholesky-based factorizations and how these can offer the potential to solve very large problems.

Key words. least-squares problems, normal equations, augmented system, sparse matrices, direct methods, iterative methods, Cholesky factorizations, preconditioning, regularization

AMS subject classifications. 65F05, 65F50

DOI. 10.1137/16M1065380

1. Introduction.

In recent years, a number of methods have been proposed for preconditioning sparse linear least-squares problems; a brief overview with a comprehensive list of references is included in the introduction to the paper of Bru et al. [4]. The recent study of Gould and Scott [20, 21] reviewed many of these methods (specifically those for which software has been made available) and then tested and compared their performance using a range of examples coming from practical applications. One of the outcomes of that study was some insight into which least-squares problems in the widely used sparse matrix collections CUTEst [19] and University of Florida [10] currently pose a real challenge for direct methods and/or iterative solvers. In particular, the study found that most of the available software packages were not reliable or efficient for rank-deficient least-squares problems (at least not when run with the recommended settings for the input parameters that were employed in the study). In this paper, we look further at such problems and focus on the effectiveness of both sparse direct solvers and iterative methods with incomplete factorization preconditioners. A key theme is the use of regularization (see, for example, [15, 48, 49]). We propose computing a factorization (either complete or incomplete) of a regularized problem and then using this as a preconditioner for an iterative solver to recover the solution of the original (unregularized) problem.

The problem we are interested in is

$$\min_x \|b - Ax\|_2, \tag{1.1}$$

*Submitted to the journal's Software and High-Performance Computing section March 11, 2016; accepted for publication (in revised form) April 17, 2017; published electronically August 24, 2017.
http://www.siam.org/journals/sisc/39-4/M106538.html
Funding: This work was supported by EPSRC grant EP/M025179/1.
†Scientific Computing Department, Rutherford Appleton Laboratory, Harwell Campus, Oxfordshire, OX11 0QX, UK, and School of Mathematical, Physical and Computational Sciences, University of Reading, Whiteknights, Reading RG6 6AQ, UK (jennifer.scott@stfc.ac.uk).

where $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is large and sparse and $b \in \mathbb{R}^m$. Our focus is on the case where A is not of full column rank. Solving (1.1) is mathematically equivalent to solving the $n \times n$ *normal equations*

$$Cx = A^T b, \qquad C = A^T A. \tag{1.2}$$

A well-known issue associated with solving (1.2) is that the condition number of the normal matrix C is the square of the condition number of A, so that the normal equations can be highly ill-conditioned [3]. Indeed, if A does not have full column rank, C is positive semidefinite and computing a Cholesky factorization breaks down: a zero (or, in practice, a negative) pivot is encountered at some stage of the factorization. In such cases, a black-box sparse Cholesky solver cannot be applied directly to (1.2). It is thus of interest to consider modifying C by adding a regularization term to allow the use of a Cholesky solver; this is explored in section 3 and compared with using a sparse symmetric indefinite solver (that incorporates numerical pivoting) for factorizing C. In particular, we look at employing the factors of the regularized normal matrix as a preconditioner for the iterative method LSMR [14] for solving (1.1).

An alternative approach is to solve the mathematically equivalent $(m+n)\times(m+n)$ *augmented system*

$$Ky = c, \qquad K = \begin{bmatrix} \gamma I_m & A \\ A^T & 0 \end{bmatrix}, \qquad y = \begin{bmatrix} \gamma^{-1} r(x) \\ x \end{bmatrix}, \qquad c = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{1.3}$$

where $\gamma > 0$, $r(x) = b - Ax$ is the residual vector, and I_m denotes the $m \times m$ identity matrix. The condition of K depends on γ and the maximum and minimum singular values of A; it varies significantly with γ, but with an appropriate choice (see [1, 3, 48]), K is much better conditioned than C. Important disadvantages of (1.3) are that K is indefinite and is generally significantly larger than the normal matrix. A sparse direct indefinite solver computes a factorization of K of the form $(PL)D(PL)^T$, where P is a permutation matrix, L is unit lower triangular, and D is block diagonal with nonsingular 1×1 and 2×2 blocks on the diagonal corresponding to 1×1 and 2×2 pivots (see, for example, [11, 27]). Using an indefinite solver may result in a more expensive (and certainly more complex) factorization process than a Cholesky solver, and, as reported in [20, 21], for large least-squares problems, the amount of memory needed may be prohibitive. One reason for this is that the analyze phase of most sparse direct solvers chooses the pivot order on the basis of the sparsity pattern of the matrix and makes the assumption that the diagonal is nonzero. When (as in the augmented system) this is not the case, it can be necessary during the subsequent numerical factorization to make significant modifications to the pivot order, leading to much higher levels of fill in the factors (entries in the factor L that were zero K) than was predicted during the analyze phase (see, for example, [28, 30]). Modifications to the pivot order are needed when a candidate pivot is found to be too small. The conditions for deciding whether a pivot is acceptable typically depend on a threshold parameter (see section 2); choosing this parameter is a compromise between retaining sparsity and ensuring stability. In section 4, we examine the effects of relaxing the threshold parameter. We also look at regularizing the problem by modifying the $(2, 2)$ block of K before performing the factorization and then using the factors as a preconditioner for an iterative solver (such as GMRES [47] or MINRES [40]) to restore accuracy in the solution of the original system.

When memory is an issue for a direct solver, an alternative approach is to use an incomplete factorization preconditioner in conjunction with an iterative solver.

Incomplete Cholesky (IC) factorizations have long been used as preconditioners for the numerical solution of large sparse, symmetric positive definite linear systems of equations; for an introduction and overview see, for example, [2, 46, 51] and the many references therein. More recently, a number of authors have considered incomplete LDL^T factorizations of symmetric quasi-definite matrices [39], saddle-point systems [52], and general indefinite systems [22, 53]. The use of a limited-memory IC factorization combined with LSMR to solve (1.1) is considered in section 5, and in section 6 we explore using incomplete LDL^T factorizations to solve (1.3). Our conclusions are drawn in section 7.

1.1. Test environment. We end this introduction by describing our test environment and test problems. The characteristics of the machine used to perform our tests are given in Table 1.1. All software is written in Fortran, and all reported timings are elapsed times in seconds. In our experiments, the direct solvers `HSL_MA87` and `HSL_MA97` (see section 2) are run in parallel, using 8 processors. We do not attempt to parallelize the sparse matrix-vector products used by the iterative solvers; moreover, the software to compute incomplete factorizations is serial. In each test, we impose a time limit of 600 seconds per problem. For the iterative methods, the number of iterations for each problem is limited to 100,000.

TABLE 1.1
Test machine characteristics.

CPU	Two Intel Xeon E5620 quadcore processors
Memory	24 GB
Compiler	gfortran version 4.8.4 with options -O3 -fopenmp
BLAS	MKL BLAS

Our test problems are taken from the CUTEst linear programme set [19] and the University of Florida Sparse Matrix Collection [10]. In each case, the matrix is "cleaned" (duplicates are summed, and out-of-range entries and explicit zeros are removed along with any null rows or columns); details of the resulting test problems are summarized in Table 1.2. Here the nullity is computed by running `HSL_MA97` on K with the pivot threshold parameter set to 0.5 (see section 2); the reported nullity is the difference between $m + n$ and the returned estimate of the matrix rank. Note that this estimate can be sensitive to the choice of ordering and scaling: `HSL_MA97` was used with no scaling and the nested dissection ordering computed by Metis [32]. In our experiments, if a right-hand side b is provided, it is used; otherwise, we take b to be the vector of 1's.

We employ the preconditioned LSMR algorithm of Fong and Saunders [14]. Like the more established LSQR algorithm [41, 42], it is based on Golub–Kahan bidiagonalization of A. However, while in exact arithmetic LSQR is mathematically equivalent to applying the conjugate gradient method to (1.2), LSMR is equivalent to applying MINRES [40], so that the quantities $\|A^T r_k\|_2$ and $\|r_k\|_2$ (where x_k and $r_k = b - Ax_k$ are the least-squares solution and residual on the kth step, respectively) are monotonically decreasing. Fong and Saunders report that LSMR may be a preferable solver because of this and because it may be able to terminate significantly earlier. Experiments in [20, 21] confirm this view and support our choice of LSMR.

Following Gould and Scott [21], in our experiments with LSMR we use the stopping rule

$$ratio(r_k) < \delta \quad \text{with} \quad ratio(r_k) = \frac{\|A^T r_k\|_2/\|r_k\|_2}{\|A^T r_0\|_2/\|r_0\|_2}. \tag{1.4}$$

TABLE 1.2

Statistics for our test set. m, n, and $nnz(A)$ are the row and column counts and the number of nonzeros in A. nullity is the estimated deficiency in the rank, density(A) is the largest ratio of the number of nonzeros in a row of A to n over all rows, density(C) is the ratio of the number of entries in C to n^2, and $\max |C_{ii}|$ and $\min |C_{ii}|$ are the largest and smallest diagonal entries in C. A "−" denotes insufficient memory to compute the statistic.

Problem	m	n	$nnz(A)$	nullity	density(A)	density(C)	$\max \lvert C_{ii}\rvert$	$\min \lvert C_{ii}\rvert$
CUTEst examples								
1. BAXTER	30733	27441	111576	2993	0.0017	0.0016	3.2×10^{5}	1.0×10^{-1}
2. DBIR1	45775	18804	1077025	103	0.0119	0.0119	8.5×10^{6}	1.0
3. DBIR2	45877	18906	1158159	101	0.0123	0.0069	3.0×10^{6}	1.0
4. LPL1	129959	39951	386218	44	0.0004	0.0003	5.4×10^{2}	1.0
5. NSCT2	37563	23003	697738	287	0.0273	0.0157	3.8×10^{6}	1.0
6. PDS-100	514577	156016	1096002	227	0.0000	0.0001	1.0	1.0
7. PDS-90	475448	142596	1014136	227	0.0000	0.0001	9.8	1.0
University of Florida Sparse Matrix Collection examples								
8. beaflw	500	492	53403	4	0.8130	0.8945	2.2×10^{5}	9.5×10^{-1}
9. 162bit	3606	3476	37118	15	0.0040	0.0195	2.5×10^{1}	7.2×10^{-3}
10. 176bit	7441	7150	82270	38	0.0022	0.0103	3.7×10^{1}	3.0×10^{-3}
11. 192bit	13691	13093	154303	81	0.0012	0.0057	5.4×10^{1}	2.5×10^{-4}
12. 208bit	24430	23191	299756	191	0.0008	0.0036	6.6×10^{-1}	1.2×10^{-4}
13. Maragal_6	21251	10144	537694	516	0.5857	0.7491	1.0×10^{1}	1.1×10^{-2}
14. Maragal_7	46845	26525	1200537	2046	0.3604	0.3099	1.3×10^{3}	1.4×10^{-2}
15. Maragal_8	60845	33093	1308415	7107	0.0503	0.0356	1.9	3.6×10^{-2}
16. mri1	114637	65536	589824	603	0.0037	0.0003	1.3	1.3
17. mri2	104597	63240	569160	−	0.0660	0.0078	1.3	1.3
18. tomographic1	59360	45908	647495	3436	0.0003	0.0009	4.4	4.5×10^{-4}

Unless indicated otherwise, we set the convergence tolerance δ to 10^{-6}. Note that (1.4) is independent of the choice of preconditioner.

When solving (1.3) using an indefinite preconditioner, we use right-preconditioned restarted GMRES [47]. Since GMRES is applied to K, a stopping criterion is applied to $Ky = c$. With the available implementations of GMRES, it is not possible during the computation to check whether the stopping condition (1.4) (which is based on A) is satisfied; it can, of course, be checked once GMRES has terminated. Instead, in our experiments involving (1.3), we use the scaled backward error

$$\frac{\|Ky_k - c\|_2}{\|c\|_2} < \epsilon, \tag{1.5}$$

where y_k is the computed solution of the augmented system on the kth step. We set the tolerance ϵ to 10^{-7}. This stopping criterion is also applied in experiments involving MINRES.

2. Sparse direct solvers. Sparse direct solvers have long been used to solve both (1.2) and (1.3). Here we briefly introduce such solvers, highlighting a number of features that are particularly relevant to understanding their performance when used for rank-deficient least-squares problems.

Sparse direct methods are designed to solve symmetric linear systems $\mathcal{A}z = f$ ($\mathcal{A} = \{\mathcal{A}_{i,j}\}$) by performing a factorization

$$\mathcal{A} = LDL^T,$$

where L is a unit lower triangular matrix and D is a block diagonal matrix with nonsingular 1×1 and 2×2 blocks. In practice, a more general factorization of the form

$$S\mathcal{A}S = (PL)D(PL)^T$$

is computed, where S is a diagonal scaling matrix and P is a permutation matrix that holds the pivot (elimination) order. If $\mathcal{A}$ is positive definite, D is diagonal with positive diagonal entries, and in this case, L may be redefined to be the lower triangular matrix $L \Leftarrow LD^{1/2}$, giving the Cholesky factorization

$$S\mathcal{A}S = (PL)(PL)^T.$$

For efficiency in terms of both time and memory it is essential to choose P to exploit the sparsity of $\mathcal{A}$. The structure of the factor L is the union of the structure of the permuted matrix $P^T\mathcal{A}P$ and new entries known as *fill*. The amount of fill is highly dependent on the choice of P. Direct solvers choose P in an analyze phase that precedes the numerical factorization and generally works solely with the sparsity pattern of $\mathcal{A}$. Observe that there is no one scaling algorithm that provides the best scaling for all possible matrices $\mathcal{A}$, and so some direct solvers offer different options for computing S, while others require (or optionally allow) the user to supply S.

For positive definite systems, the chosen pivot order can be used unaltered by the numerical factorization. However, for indefinite systems, it may be necessary to make modifications to maintain numerical stability. This is done by delaying the elimination of variables that could cause instability until later in the factorization when the associated pivot (that is, the 1×1 or 2×2 block used to eliminate one, respectively, two variables) can be safely used. The exact method used to select pivots during the numerical factorization varies from solver to solver, but essentially each seeks to avoid dividing a large off-diagonal entry by a small diagonal one. If the elimination of variable k is delayed, either an update from another elimination will increase the magnitude of the diagonal entry $\mathcal{A}_{k,k}$, or column k will become adjacent to column $k+1$ with the property that $\mathcal{A}_{k,k+1}$ is large and hence can be incorporated into a stable 2×2 pivot.

There are several modern sparse direct solvers available for solving positive definite problems. Some are designed exclusively for such systems (for example, CHOLMOD [6] and `HSL_MA87` [25]), while others can also be used to solve indefinite systems (notably, `MA57` [11], `HSL_MA97` [27], MUMPS [38], WSMP [23], PARDISO [43], and `SPRAL_SSIDS` [24]). We employ the packages `HSL_MA87` and `HSL_MA97` from the HSL Mathematical Software Library [31]; an overview of both packages together with a numerical comparison is provided by Hogg and Scott [29].

The Cholesky solver `HSL_MA87` is designed to run on multicore architectures. It splits each part of the computation into tasks of modest size but sufficiently large that good level-3 BLAS performance can be achieved. The dependencies between the tasks are implicitly represented by a directed acyclic graph (DAG). This solver requires the user to supply both the pivot order P and the scaling S (other HSL packages are available for computing these).

By contrast, `HSL_MA97` is a parallel multifrontal code that is able to solve both positive definite and indefinite systems (although its performance on positive definite systems is generally not competitive with that of `HSL_MA87`). `HSL_MA97` offers a range of ordering and scaling options and also allows the user to supply P and/or S. In a multifrontal method, the factorization of $\mathcal{A}$ proceeds using a succession of assembly operations of small dense matrices, interleaved with partial factorizations of these matrices. The assembly operations can be recorded as a tree, known as an assembly tree. The assembly proceeds from the leaf nodes up the tree to the root node(s). Typically, most of the flops are performed at the root node and the final few levels of the tree. For the efficient and stable partial factorization of the dense submatrices, `HSL_MA97` uses separate computational kernels for the positive definite and indefinite

cases. For the latter, HSL_MA97 employs the sufficient (but not necessary) conditions given by Duff et al. [12] for threshold partial pivoting to be stable. Let $\mathcal{A}^{(k)}$ denote the Schur complement after columns $1,\ldots,k-1$ of $\mathcal{A}$ have been eliminated, and let $u \in [0, 0.5]$ be the pivot threshold. The criteria for stability are the following:

- A 1×1 pivot on column k is stable if

$$\max_{i>k} |\mathcal{A}^{(k)}_{i,k}| < u^{-1} |\mathcal{A}^{(k)}_{k,k}|. \tag{2.1}$$

- A 2×2 pivot on columns k and $k+1$ is stable if

$$\left| \begin{pmatrix} \mathcal{A}^{(k)}_{k,k} & \mathcal{A}^{(k)}_{k,k+1} \\ \mathcal{A}^{(k)}_{k+1,k} & \mathcal{A}^{(k)}_{k+1,k+1} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{i>k+1} |\mathcal{A}^{(k)}_{i,k}| \\ \max_{i>k+1} |\mathcal{A}^{(k)}_{i,k+1}| \end{pmatrix} \le u^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tag{2.2}$$

  where the modulus of the matrix is interpreted elementwise. Additionally, it is required that the pivot be nonsingular and inverted stably.

In the case where u is zero, this is interpreted as requiring that the pivot be nonsingular. Observe that these conditions imply that each entry in L is bounded in modulus by u^{-1}. The choice of u is a compromise between stability and sparsity: the larger u is, the more stable the factorization will be, but the fill in L may increase as more pivots may fail the stability test, causing them to be delayed. The default value within HSL_MA97 is $u = 0.01$, and, unless stated otherwise, this value is used in our experiments. Note that including pivoting in an indefinite sparse direct solver is necessary for stability, but it has the major disadvantage of hindering parallelism. Note also that HSL_MA97 is designed to handle the case where the system matrix $\mathcal{A}$ is singular. The code holds D^{-1}, and when a zero pivot is encountered, the corresponding entry of D^{-1} is set to zero. The corresponding components of the solution vector are thus set to zero.

Both HSL_MA87 and HSL_MA97 employ a technique known as node amalgamation. This has become well established as a means of improving the factorization speed at the expense of the number of entries in the factor L and the operation counts during the factorization and subsequent solve phase. During the analyze phase, a child node in the tree is merged with its parent if both parent and child have fewer than a prescribed number nemin of variables that are eliminated, or if merging parent and child generates no additional nonzeros in L. The value of the parameter nemin determines the level of node amalgamation, with a value in the range of 8 to 32 typically recommended as providing a good balance between sparsity and efficiency in the factorize and solve phases (see [25, 44]). In our experiments, we set nemin equal to 32.

3. Solving the normal equations with a sparse direct solver.

3.1. Using a direct solver as a preconditioner for LSMR. If A does not have full column rank, $C = A^T A$ is symmetric and positive semidefinite, and thus attempting to compute a Cholesky factorization will suffer breakdown. Breakdown happens when a zero (or negative) pivot is encountered; if this happens, a Cholesky solver will terminate the computation with an error flag. In this section, we consider using a regularization term to allow a Cholesky factorization to be used and compare this approach with employing a symmetric indefinite solver to factorize C. The former requires an iterative method to recover the required accuracy in the solution, while the latter involves the overhead of pivoting; numerical results are used to explore which approach is the most efficient.

When C is positive semidefinite and breakdown of a Cholesky factorization occurs, a simple remedy is to employ a global shift $\alpha > 0$ and then compute a Cholesky factorization of the scaled and shifted matrix

$$C_\alpha = SA^TAS + \alpha I. \tag{3.1}$$

Here S is again a diagonal scaling matrix. The shift α is also referred to as a *Tikhonov regularization* parameter. The choice of α should be related to the smallest eigenvalue of SA^TAS, but this information is not readily available. Clearly it is always possible to find an α so that C_α is positive definite; if the initial choice α is too small (that is, C_α is positive semidefinite), it may be necessary to restart the factorization more than once, increasing α on each restart until breakdown is avoided. If the regularized normal equations

$$C_\alpha x_\alpha = SA^Tb, \qquad x = Sx_\alpha, \tag{3.2}$$

are solved, the computed value of the least-squares objective $\|r_\alpha\|_2 = \|b - ASx_\alpha\|_2$ may differ from the optimum for the original problem. We can seek to obtain the solution x to (1.1) by applying a refinement process. The standard approach for linear systems is to employ a small number of steps of iterative refinement. However, iterative refinement is not effective when applied to the normal equations if the normal matrix is ill-conditioned [3]. We thus propose using the Cholesky factors $L_\alpha L_\alpha^T$ of C_α as a preconditioner for LSMR applied to the original (unshifted) problem.

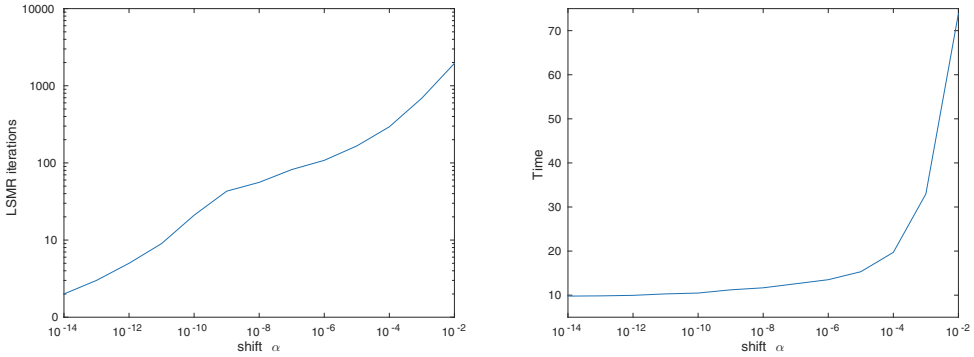

FIG. 3.1. *The effect of the shift α on the number of LSMR iterations (left) and the time (right) for problem* `Maragal_6`. `HSL_MA87` *is used to compute a preconditioner for LSMR.*

In our experiments, we use l_2-norm scaling of A, that is, $S_{ii} = 1/\|Ae_i\|_2$ (where e_i is the ith unit vector) so that each column of A is normalized by its 2-norm. The diagonal entries of SA^TAS are then all 1. In the positive definite case, van der Sluis [55] proved that if all the diagonal entries of SCS are equal, then it has condition number close to the optimal among diagonal scalings of C. In Figure 3.1, we plot the number of iterations required by preconditioned LSMR and the total time for solving problem `Maragal_6` using values of the shift α in the range 10^{-14} to 10^{-2}; here L_α is computed using the positive definite solver `HSL_MA87` with Metis [32] nested dissection ordering. Similar patterns are observed for other problems, although for some of our test examples (including DBIR2 and mri2) we found with the l_2-norm scaling of A that we needed to use $\alpha \geq 10^{-12}$ to avoid breakdown. In Table 3.1, we report results

TABLE 3.1

Results for solving the least-squares problem using the Cholesky solver HSL_MA87 *to compute a preconditioner for LSMR. The shift is* $\alpha = 10^{-12}$. $nnz(L_\alpha)$ *denotes the number of entries in the* HSL_MA87 *factor, time is the total solution time (in seconds), and itn is the number of LSMR iterations. The value of the least-squares objective before and after applying LSMR is* $\|r_\alpha\|_2$ *and* $\|r\|_2$*, respectively. Results are also given for* HSL_MA97 *run in indefinite mode (with no shift).*

Problem	HSL_MA87 (positive definite)					HSL_MA97 (indefinite)		
	$nnz(L_\alpha)$	*time*	$\|r_\alpha\|_2$	$\|r\|_2$	*itn*	$nnz(L)$	*time*	$\|r\|_2$
BAXTER	6.83×10^6	0.35	6.683×10^1	5.929×10^1	6	7.02×10^6	0.48	5.929×10^1
DBIR1	4.27×10^6	0.71	1.667×10^2	1.667×10^2	1	4.59×10^6	0.89	1.667×10^2
DBIR2	4.94×10^6	0.79	1.665×10^2	1.665×10^2	1	4.86×10^6	0.97	1.665×10^2
LPL1	7.44×10^6	0.31	7.088×10^1	7.088×10^1	1	7.94×10^6	0.38	7.088×10^1
NSCT2	8.81×10^6	1.54	1.838×10^2	1.838×10^2	1	8.42×10^6	1.88	1.838×10^2
PDS-100	5.91×10^7	2.03	2.849×10^2	2.849×10^2	1	5.79×10^7	2.11	2.849×10^2
PDS-90	5.23×10^7	1.79	2.685×10^2	2.685×10^2	1	5.14×10^7	2.27	2.685×10^2
beaflw	1.17×10^5	0.04	4.376	4.305	7	1.15×10^5	0.05	4.200
162bit	2.82×10^6	0.10	1.177×10^1	1.177×10^1	1	2.85×10^6	0.28	1.177×10^1
176bit	1.02×10^7	0.37	1.842×10^1	1.842×10^1	1	1.03×10^7	1.73	1.842×10^1
192bit	2.85×10^7	1.37	2.485×10^1	2.485×10^1	1	2.97×10^7	7.55	2.485×10^1
208bit	8.60×10^7	6.81	3.850×10^1	3.850×10^1	1	8.54×10^7	35.1	3.850×10^1
Maragal_6	4.96×10^7	10.8	1.069×10^1	1.069×10^1	3	5.06×10^7	17.8	1.069×10^1
Maragal_7	1.43×10^8	30.6	1.369×10^1	1.369×10^1	3	1.39×10^8	40.8	1.369×10^1
Maragal_8	8.85×10^7	9.57	2.383×10^2	2.378×10^2	27	9.88×10^7	25.3	2.379×10^2
mri1	8.27×10^6	0.50	2.674×10^1	2.674×10^1	1	8.69×10^6	0.63	2.674×10^1
mri2	3.43×10^7	2.65	1.413×10^2	1.413×10^2	1	3.78×10^7	4.94	1.413×10^2
tomographic1	2.96×10^7	1.20	4.185×10^1	4.185×10^1	3	3.20×10^7	2.37	4.185×10^1

for $\alpha = 10^{-12}$. In many cases, the requested accuracy is achieved with a single step of LSMR. We did experiment with running HSL_MA97 in positive definite mode, but, as reported in [29], it is generally slower than HSL_MA87, and so detailed results are omitted.

3.2. Comparison with using a general indefinite solver. An alternative to computing a Cholesky factorization of the regularized normal equations and using the factor to precondition LSMR is to solve the original (unshifted) normal equations using a sparse symmetric indefinite direct solver that allows the system matrix to be singular, provided that the equations to be solved are consistent; this is always true for the normal equations. The latter has the advantage of not requiring the selection of a shift α, but allowing pivoting for numerical stability (as discussed in section 2) adds to the factorization cost. Results for our test problems run with the solver HSL_MA97 in indefinite mode are included in columns 7–9 of Table 3.1 (using the l_2-norm scaling of A, Metis ordering, and default threshold parameter $u = 0.01$). No refinement is used. We see that the positive definite solver applied to the scaled and regularized normal equations results in a faster total solution time compared to the indefinite approach.

4. Solving the augmented system with a direct solver. In this section, we look at using a direct solver to solve the augmented system (1.3). We first consider employing a general-purpose symmetric indefinite solver that incorporates pivoting. We report on the effects of the choice of scaling on the fill in the factors and computation time and show that either a matching-based scaling or l_2-norm scaling of A is generally the method of choice. We also examine whether the tactic that is sometimes used in optimization problems of reducing the threshold parameter u can improve the solver performance without resulting in instability. Then, in section 4.2, we consider

regularizing the augmented system. The factors of the regularized matrix are used to provide a preconditioner for an iterative method. Numerical results are compared with those for the regularized normal equations presented in the last section.

4.1. Using a direct solver for $Ky = c$. Consider applying HSL_MA97 to the augmented system (1.3). The computed factorization is

$$SKS = (PL)D(PL)^T,$$

with S a diagonal scaling matrix, P a permutation matrix, and D block diagonal (with 1×1 and 2×2 blocks). During the factorization of K, the zero $(2,2)$ block can lead to many modifications being made to the analyze pivot order to preserve stability; this is reflected in the number of delayed pivots reported by HSL_MA97 ($ndelay$). It is well known that for some sparse indefinite problems, the choice of the scaling S can have a significant impact on reducing the number of delayed pivots and hence the fill in L and overall performance (see, for example, [26]). When developing HSL_MA97, Hogg and Scott [30] studied pivoting strategies for tough sparse indefinite systems. The scaling options offered by HSL_MA97 (1) generate a scaling using a weighted bipartite matching (MC64) [13]; or (2) generate an equilibration-based scaling (MC77) [45]; or (3) use a matching-based ordering and the scaling that is generated as a side effect of this process (MC80); or (4) generate a scaling by minimizing the absolute sum of log values in the scaled matrix (MC30). Note that these scalings do not exploit the block structure of K but treat K as a general indefinite sparse symmetric matrix. In addition to the built-in HSL_MA97 options, we compute and input the l_2-norm scaling of K and compute the l_2-norm scaling of A. In the latter case, we input the scaling matrix

$$S = \begin{bmatrix} I_m & \\ & \tilde{S} \end{bmatrix}, \tag{4.1}$$

where $\tilde{S}_{ii} = 1/\|Ae_i\|_2$. We illustrate the effects of scaling on a subset of our problems in Table 4.1. Here we use the default threshold parameter $u = 0.01$ and Metis ordering and set the parameter $\gamma = 1$ in (1.3). These examples were chosen because they were found to be sensitive to the scaling; for some of our other test examples, it had a much smaller effect. Closer examination shows that if we compute the diagonal entries of C, the ratio of the largest to the smallest diagonal entry for these examples is large (see Table 1.2), indicating poor initial scaling. As expected, no single scaling gives the best results on all problems. The matching-based MC64 scaling can lead to sparse factors, but it can be expensive to compute. Based on our findings, we use l_2-norm scaling of A for our remaining experiments with HSL_MA97 applied to K, and we set $\gamma = 1$ [1].

Results for solving the augmented system are given in Table 4.2. Compared to employing a direct solver to factorize the normal equations (Table 3.1), we see that, in general, factorizing the augmented system with the default threshold parameter $u = 0.01$ results in significantly more entries in the factors plus slower computation times. In some optimization applications (see, for example, [17, 49]), it is common practice to try to make the factorization of indefinite systems more efficient in terms of time and memory by employing a relaxed threshold parameter u and to increase u only if the linear system is not solved with sufficient accuracy. Thus in Table 4.2 we also include results for $u = 10^{-8}$. With the exception of problems BAXTER and NSCT2, this gives only a modest reduction in the number of entries in the factors and

TABLE 4.1

The effects of scaling on the performance of HSL_MA97 *for solving the augmented system* (1.3) *($u = 0.01$, Metis ordering, and $\gamma = 1$). $nnz(L)$ denotes the number of entries in the factor, ndelay is the number of delayed pivots, and time is the total solution time (in seconds). $l_2(K)$ and $l_2(A)$ denote l_2-norm scaling of K and A, respectively. For each problem, the sparsest factors and fastest times are in bold.*

Problem	Scaling	$nnz(L)$	*ndelay*	*time*
BAXTER	none	8.36×10^6	2.74×10^4	0.50
	MC30	1.62×10^6	5.66×10^3	**0.24**
	MC64	$\mathbf{1.13}\times10^6$	1.08×10^3	0.26
	MC77	2.12×10^6	6.00×10^3	0.29
	MC80	1.49×10^6	1.37×10^2	0.27
	$l_2(K)$	5.72×10^6	2.00×10^4	0.38
	$l_2(A)$	1.38×10^7	3.41×10^4	0.73
DBIR1	none	9.79×10^7	1.73×10^5	47.6
	MC30	1.16×10^8	1.88×10^5	52.6
	MC64	9.02×10^6	5.02×10^3	1.44
	MC77	1.58×10^8	2.33×10^5	140
	MC80	2.03×10^7	3.93×10^4	3.62
	$l_2(K)$	1.05×10^8	1.80×10^5	62.1
	$l_2(A)$	$\mathbf{8.76}\times10^6$	1.12×10^3	**1.15**
DBIR2	none	1.23×10^8	1.23×10^5	26.6
	MC30	2.68×10^7	3.37×10^4	3.79
	MC64	$\mathbf{8.54}\times10^6$	4.00×10^2	1.41
	MC77	1.33×10^8	1.27×10^5	61.2
	MC80	1.96×10^7	1.12×10^4	3.17
	$l_2(K)$	1.00×10^8	1.04×10^5	18.5
	$l_2(A)$	8.85×10^6	9.62×10^2	**1.14**

TABLE 4.2

Results for solving the augmented system (1.3) *using the direct solver* HSL_MA97 *with threshold parameter u set to 0.01 and 10^{-8}. $nnz(L)$ denotes the number of entries in the factor, ndelay is the number of delayed pivots, time is the total solution time (in seconds), and the value of the least-squares objective is $\|r\|_2$.*

Problem	$u = 0.01$				$u = 10^{-8}$			
	$nnz(L)$	*ndelay*	*time*	$\|r\|_2$	$nnz(L)$	*ndelay*	*time*	$\|r\|_2$
BAXTER	1.38×10^7	3.41×10^4	0.73	5.929×10^1	1.49×10^6	4.75×10^3	0.22	5.929×10^1
DBIR1	8.76×10^6	1.12×10^3	1.15	1.667×10^2	8.75×10^6	1.08×10^3	1.13	1.667×10^2
DBIR2	8.85×10^6	9.62×10^2	1.13	1.665×10^2	8.46×10^6	1.11×10^2	1.12	1.665×10^2
LPL1	1.45×10^7	4.34×10^3	1.02	7.088×10^1	1.38×10^7	3.0	0.97	7.088×10^1
NSCT2	1.07×10^7	5.73×10^3	0.79	1.838×10^2	7.60×10^6	1.05×10^3	0.69	1.838×10^2
PDS-100	1.05×10^8	0.0	6.67	2.849×10^2	1.05×10^8	0.0	6.61	2.849×10^2
PDS-90	1.00×10^8	0.0	8.06	2.685×10^2	1.00×10^8	0.0	8.07	2.685×10^2
beaflw	2.65×10^5	2.95×10^2	0.05	4.162	2.51×10^5	6.70×10^1	0.03	4.162
162bit	4.25×10^6	3.42×10^2	0.33	1.177×10^1	4.23×10^6	1.59×10^2	0.31	1.177×10^1
176bit	1.52×10^7	2.33×10^3	2.07	1.842×10^1	1.50×10^7	1.07×10^3	1.98	1.842×10^1
192bit	4.51×10^7	9.00×10^3	7.85	2.485×10^1	4.44×10^7	3.80×10^3	7.59	2.485×10^1
208bit	1.30×10^8	2.45×10^4	44.1	3.850×10^1	1.28×10^8	1.09×10^4	42.1	3.850×10^1
Maragal_6	2.30×10^7	4.18×10^4	3.02	1.069×10^1	2.30×10^7	4.17×10^4	2.75	1.069×10^1
Maragal_7	7.03×10^7	3.48×10^4	9.32	1.369×10^1	7.03×10^7	3.47×10^4	8.41	1.369×10^1
Maragal_8	1.75×10^8	2.92×10^5	89.2	2.378×10^2	1.77×10^8	2.89×10^5	58.0	2.379×10^2
mri1	1.71×10^7	1.70×10^4	1.42	2.674×10^1	1.65×10^7	1.48×10^4	1.33	2.674×10^1
mri2	1.48×10^8	1.79×10^5	74.7	1.413×10^2	1.44×10^8	1.75×10^5	38.7	1.413×10^2
tomographic1	5.10×10^7	7.62×10^4	5.98	4.185×10^1	4.68×10^7	6.00×10^4	4.78	4.185×10^1

in the number of delayed pivots, but there can be a significant reduction in the time (including problems Maragal_8 and mri2). Further examination reveals that pivots are still being rejected at the nonroot nodes and passed up the assembly tree to the

root node, where a dense factorization is performed. The delayed pivots result in the root node being much larger than was predicted by the analyze phase, and, in this case, the factorization of the root node accounts for most of the operations and time. Using a small value of the threshold parameter significantly reduces the time for the root node factorization, and it is this that leads to the overall reduction in the time.

4.2. Regularized augmented system. To attempt to reduce the number of entries in the factors of the augmented system resulting from delayed pivots, we consider the regularized system

$$K_\beta y_\beta = c, \qquad K_\beta = \begin{bmatrix} I_m & A \\ A^T & -\beta I_n \end{bmatrix}, \qquad y_\beta = \begin{bmatrix} r(x_\beta) \\ x_\beta \end{bmatrix}, \qquad c = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{4.2}$$

where $r(x_\beta) = b - Ax_\beta$ and $\beta > 0$ (see, for example, [16, 48]). This is a symmetric quasi-definite (SQD) system. Vanderbei [56] shows that, in exact arithmetic, SQD systems are strongly factorizable, i.e., a signed Cholesky factorization of the form $\mathrm{LDL}^{\mathrm{T}}$ (with D diagonal having both positive and negative entries) exists for any symmetric permutation P. Thus P can be chosen to maintain sparsity. However, the signed Cholesky factorization may be unstable. A stability analysis is given by Gill, Saunders, and Shinnerl [17] (see also [15, 18]), which shows the importance of the effective condition number of K_β for the stability of the factorization.

We note that other regularizations of the augmented system have been proposed. In particular, Saunders [48, 49] and George and Saunders [15] use the SQD matrix

$$K_{\beta_1\beta_2} = \begin{bmatrix} \beta_1 I_m & A \\ A^T & -\beta_2 I_n \end{bmatrix},$$

with $\beta_1, \beta_2 > 0$, and in their experiments they set $\beta_1 = \beta_2 = 10^{-6}$. Saunders suggests that this may be favorable when A is ill-conditioned.

We apply our sparse symmetric indefinite solver `HSL_MA97` to the scaled regularized augmented matrix $SK_\beta S$ with the threshold parameter u set to 0.0 and S given by (4.1). With $\beta > 0$, the computed value of the least-squares objective $\|r_\beta\|_2 = \|b - A\tilde{S}x_\beta\|_2$ may differ from the optimum for the original problem, and if the stopping criterion (1.4) is not satisfied, we propose using the factors as a preconditioner for an iterative method [15]. Here we use right-preconditioned GMRES applied to the original augmented system (1.3) (with $\gamma = 1$). Here GMRES is used without restarting, as only a small number of iterations are required. An alternative would be to use the symmetric solver MINRES [40]. This requires a positive definite preconditioner, and so we could employ the method presented by Gill et al. [16] to modify D_β and L_β (note that this approach has been used recently by Greif, He, and Liu [22]).

Our results using GMRES are given in Figure 4.1 and Table 4.3. The figure looks at the effects of varying the regularization parameter β on the number of iterations and the solution time for problem `Maragal_6`. As β increases, so too do the iteration count and time; similar patterns are seen for our other test examples. For the results in Table 4.3, we set $\beta = 10^{-8}$. For our test set, this gives no delayed pivots. While the precise choice of β is not important, if β is "too small," some pivots may get delayed and the factorization become less stable, resulting in more GMRES iterations being needed for the requested accuracy than for a larger β. Comparing Tables 4.2 and 4.3, we note that we obtain much sparser factors than previously, and, as the number of iterations of GMRES is generally modest (indeed, often we did not need

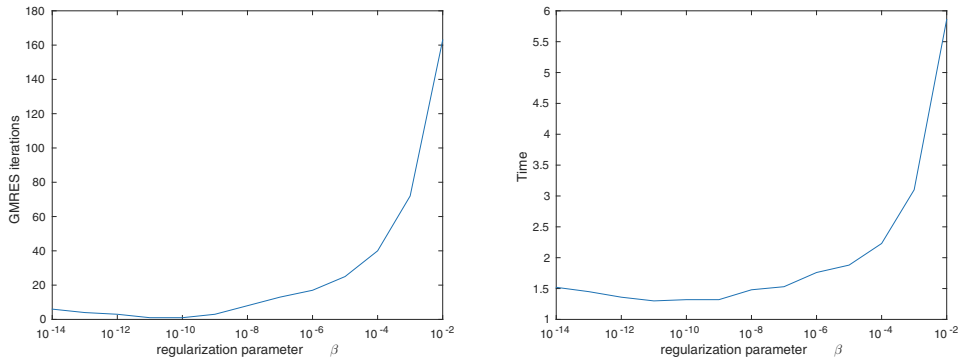

FIG. 4.1. *The effect of the regularization parameter β on the number of GMRES iterations (left) and the time (right) for problem* `Maragal_6` *(threshold $u = 0.0$).*

TABLE 4.3

Results for solving the regularized augmented (SQD) system with $\beta = 10^{-8}$ using the direct solver `HSL_MA97` *with threshold parameter $u = 0.0$. $nnz(L_\beta)$ denotes the number of entries in the factor, $time_f$ and $time_t$ are the* `HSL_MA97` *and total solution times (in seconds), the value of the least-squares objective before and after applying preconditioned GMRES is $\|r_\beta\|_2$ and $\|r\|_2$, respectively, and itn is the number of GMRES iterations.*

Problem	$nnz(L_\beta)$	$time_f$	$time_t$	$\|r_\beta\|_2$	$\|r\|_2$	itn
`BAXTER`	1.07×10^6	0.22	1.84	7.496×10^1	5.929×10^1	234
`DBIR1`	8.38×10^6	1.12	1.16	1.667×10^2	1.667×10^2	1
`DBIR2`	8.44×10^6	1.11	1.12	1.665×10^2	1.665×10^2	0
`LPL1`	1.38×10^7	0.97	0.97	7.088×10^1	7.088×10^1	0
`NSCT2`	7.38×10^6	0.67	0.67	1.838×10^2	1.838×10^2	0
`PDS-100`	1.05×10^8	6.64	6.66	2.849×10^2	2.849×10^2	0
`PDS-90`	1.00×10^8	8.18	8.20	2.685×10^2	2.685×10^2	0
`beaflw`	2.32×10^5	0.04	0.04	4.180	4.162	4
`162bit`	4.16×10^6	0.29	0.31	1.179×10^1	1.177×10^1	2
`176bit`	1.46×10^7	1.95	2.07	1.844×10^1	1.842×10^1	4
`192bit`	4.32×10^7	7.30	7.54	2.489×10^1	2.485×10^1	3
`208bit`	1.25×10^8	41.2	42.7	3.865×10^1	3.850×10^1	8
`Maragal_6`	1.50×10^7	1.28	1.50	1.069×10^1	1.069×10^1	8
`Maragal_7`	2.29×10^7	2.27	2.27	1.369×10^1	1.369×10^1	0
`Maragal_8`	2.31×10^7	3.74	4.62	2.388×10^2	2.386×10^2	17
`mri1`	1.34×10^7	1.00	1.01	2.674×10^1	2.674×10^1	0
`mri2`	8.72×10^6	1.06	1.08	1.413×10^2	1.413×10^2	0
`tomographic1`	3.16×10^7	2.70	3.08	4.206×10^1	4.194×10^1	5

to use GMRES to obtain the requested accuracy), we have significantly faster total solution times (note, in particular, problems `Maragal_8` and `mir2`). We observe that for problem `BAXTER`, we can reduce the number of GMRES iterations if we use a smaller β: with $\beta = 10^{-11}$, only 11 iterations are needed.

The number of entries in the factors and the total solution times using `HSL_MA97` to solve the regularized augmented system (4.2) and `HSL_MA87` applied to the regularized normal equations are compared in Figure 4.2. A point above the line $y = 1$ indicates that using the normal equations is the better choice; the converse is true for a point below the line. We see that in many cases there is little to choose between the approaches in terms of the size of the factors, but that for a small number of examples (including the `Maragal` problems) the augmented system factors are significantly

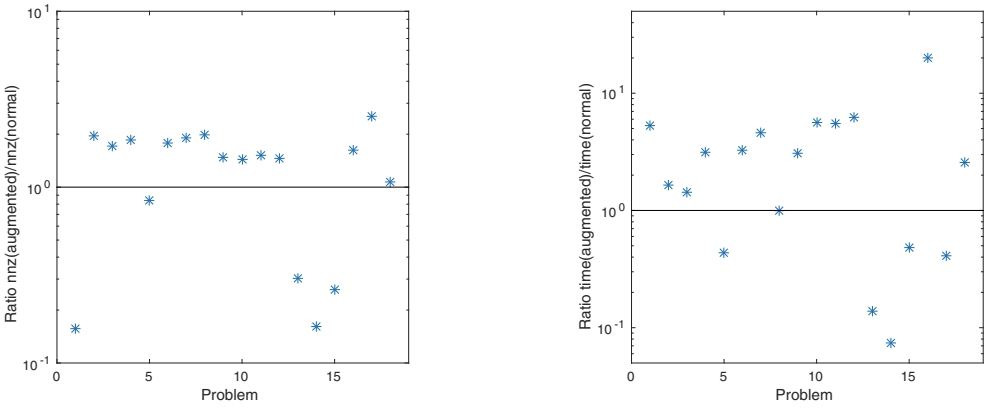

FIG. 4.2. *Ratios of the number of entries in the factor (left) and the time (right) for the regularized augmented system solved using* `HSL_MA97` *in indefinite mode and the regularized normal equations solved using the positive definite solver* `HSL_MA87`.

sparser. However, the normal equation approach with the positive definite solver is faster for most of the remaining problems.

5. Incomplete factorization of the normal matrix C. Having explored in section 3 complete factorizations of the (regularized) normal equations, in this section we look at computing an incomplete Cholesky (IC) factorization for use as a preconditioner for LSMR. An IC factorization takes the form LL^T in which some of the fill entries that would occur in a complete factorization are ignored. The preconditioned normal equations become

$$(AL^{-T})^T(AL^{-T})z = L^{-1}CL^{-T}z = L^{-1}A^Tb, \qquad z = L^Tx.$$

Performing preconditioning operations involves solving triangular systems with L and L^T. Over the years, a wealth of different IC variants have been proposed, including structure-based methods, those based on dropping entries below a prescribed threshold, and those based on prescribing the maximum number of entries allowed in L (see, for instance, [2, 46, 51] and the references therein). Level-based methods (IC(k)) that are based on the sparsity pattern of A plus a small number of levels of fill are popular and straightforward to implement. However, they are best suited to linear systems arising from discretized partial differential equations. Here we use the limited-memory approach of Scott and Tůma [50, 51], which generalizes the ICFS algorithm of Lin and Moré [35]. The scheme is based on a matrix factorization of the form

$$C = (L + R)(L + R)^T - E, \tag{5.1}$$

where L is the lower triangular matrix with positive diagonal entries that is used for preconditioning, R is a strictly lower triangular matrix with small entries that is used to stabilize the factorization process but is subsequently discarded, and $E = RR^T$. On the jth step of the factorization, the first column of the Schur complement is decomposed into a sum of two vectors $l_j + r_j$, such that $|l_j|^T|r_j| = 0$ (with the first entry in l_j nonzero), where l_j (respectively, r_j) contains the entries that are retained in (respectively, discarded from) the incomplete factorization. In the next step of a complete decomposition, the Schur complement of order $n - j$ would be updated by subtracting the outer product of the pivot row and column, that is,

by subtracting $(l_j + r_j)(l_j + r_j)^T$. In the incomplete case, the positive semidefinite term $E_j = r_j r_j^T$ is not subtracted. Moreover, to further limit the memory required, drop tolerances are (optionally) used. If at some stage a zero or negative pivot is encountered, the factorization suffers breakdown and, as in section 3, a shift is applied and the incomplete factorization of the shifted matrix (3.1) is computed.

A software package `HSL_MI35` that implements this limited-memory IC algorithm for least-squares problems has been developed. This code is a modification of `HSL_MI28` [50], which is designed for symmetric positive definite systems. Modifications were needed to allow the user to specify the maximum number of entries allowed in each column of the incomplete factor L (in `HSL_MI28` the user specified the amount of fill allowed, but as columns of the normal matrix C may be dense, or close to dense, this change was needed to keep L sparse). Furthermore, there is no need to form and store all of C explicitly; rather, the lower triangular part of its columns can be computed one at a time and then used to perform the corresponding step of the incomplete Cholesky algorithm before being discarded. `HSL_MI35` includes a number of scaling and ordering options so that an incomplete factorization of

$$\overline{C}_\alpha = P^T S C S P + \alpha I$$

is computed, where P is a permutation matrix chosen on the basis of sparsity, S is a diagonal scaling matrix, and $\alpha \geq 0.0$. Based on extensive experimentation in [50], the default ordering is the profile reduction ordering of Sloan [54]. We experimented with using the l_2-norm scaling of A and also, as discussed in section 3, applying the l_2-norm scaling of A, forming $C_S = SA^T AS$, and then applying the l_2-norm scaling of C_S (in practice, this requires us to compute only one column of C_S at a time). We found that for some examples this "double" scaling resulted in a smaller shift α being needed and hence gave a higher quality preconditioner. Thus we use this for our reported results. In the following, *lsize* and *rsize* denote the parameters that control the maximum number of entries in each column of L and R, respectively. In each test, the initial shift α is 0.001. In the event of breakdown, it is increased until the incomplete factorization is successful (see [50] for details). The recorded times include the time to restart the factorization following any breakdowns.

Figure 5.1 illustrates the potential benefits of employing intermediate memory R in the construction of L (note that the ICFS software [35] employs no intermediate memory). Here we set $lsize = 200$ and vary *rsize* from 0 to 1000; the drop tolerances are set to 0.0. For each value of *rsize*, the number of entries in L is $nnz(L) = 6.63 \times 10^6$. We see that increasing the intermediate memory stabilizes the factorization, reducing the shift and giving a higher quality preconditioner that requires fewer LSMR iterations and less time. Observe that because the number of restarts following a breakdown decreases as *rsize* increases, the time for computing the IC factorization does not necessarily increase with *rsize*.

In Table 5.1, we report results for `HSL_MI35` applied to our test set; default settings are used for the ordering and dropping parameters. We experimented with a range of values for the parameters *lsize* and *rsize* that control the memory, and for each example we chose values that perform well. While the preconditioner quality improves and the number of LSMR iterations decreases as the memory increases, the factorization times generally increase and, because $nnz(L)$ increases, each application of the preconditioner becomes more expensive. Thus the best values of *lsize* and *rsize* in terms of the total time are highly problem dependent (in particular, we found that using $rsize > 0$ is not always beneficial); the results given in Table 5.1 illustrate this. We struggle to solve problem `BAXTER`: a large number of iterations are required, and

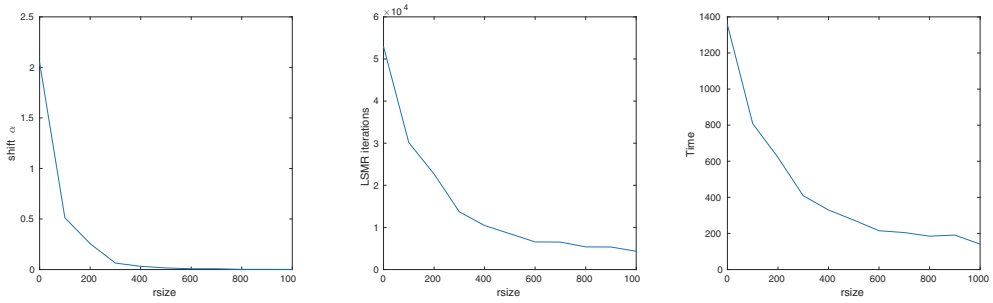

FIG. 5.1. *The effect of increasing the amount of intermediate memory used in the construction of the* HSL_MI35 *IC preconditioner on the size of the shift (left), the number of LSMR iterations (center), and the total time in seconds (right) for problem* Maragal_8.

TABLE 5.1

Results for LSMR with the IC factorization preconditioner from HSL_MI35 *applied to* $C = A^T A$. *lsize and rsize control the memory used by* HSL_MI35, $nnz(L)$ *denotes the number of entries in the factor* L, α *is the shift,* $time_f$ *and* $time_t$ *are the factorization and total solution times (in seconds), the value of the least-squares objective is* $\|r\|_2$, *and the number of LSMR iterations is itn.*

Problem	*lsize*	*rsize*	$nnz(L)$	α	$time_f$	$time_t$	$\|r\|_2$	*itn*
BAXTER	100	100	5.35×10^5	0.001	0.14	105	5.990×10^1	42410
DBIR1	100	0	5.05×10^5	0.002	0.50	0.68	1.667×10^2	40
DBIR2	100	0	5.34×10^5	0.002	0.55	0.79	1.665×10^2	50
LPL1	100	0	9.15×10^5	0.001	0.15	0.54	7.088×10^1	70
NSCT2	100	0	8.28×10^5	0.002	1.23	1.40	1.838×10^2	40
PDS-100	20	20	2.89×10^6	0.001	1.09	3.18	2.849×10^2	90
PDS-90	20	20	2.62×10^6	0.001	0.99	2.90	2.685×10^2	90
beaflw	100	200	4.01×10^4	0.016	0.07	1.89	4.533	9570
162bit	20	20	7.04×10^4	0.001	0.05	0.14	1.177×10^1	220
176bit	20	20	1.46×10^5	0.001	0.12	0.48	1.842×10^1	440
192bit	20	20	2.66×10^5	0.004	0.41	2.08	2.485×10^1	1070
208bit	20	20	4.69×10^5	0.002	0.64	4.92	3.850×10^1	1430
Maragal_6	20	20	2.12×10^5	0.256	6.43	7.59	1.069×10^1	590
Maragal_7	100	0	2.56×10^6	0.256	17.6	19.5	1.369×10^1	170
Maragal_8	200	1000	1.39×10^6	0.001	13.8	59.8	2.387×10^2	5110
mri1	100	100	1.10×10^6	0.001	0.55	1.41	2.674×10^1	100
mri2	20	20	7.83×10^5	2.048	3.13	18.4	1.413×10^2	2530
tomographic1	100	0	3.29×10^6	0.001	0.78	8.74	4.192×10^1	590

the computed $\|r\|_2$ is (slightly) larger than reported elsewhere. However, if we compare the results for the other examples with those in Table 3.1 for the direct solver applied to the normal equations, we see that for many examples, the IC times are competitive with the direct solver times. Moreover, the IC factorization produces significantly sparser factors, giving it the potential to be used successfully for much larger problems than can be tackled by a direct solver. Observe that the shift α for the IC factorization is significantly larger than that used by the direct solver. Consequently, the number of iterations needed for convergence can be large.

6. Preconditioning strategies for the augmented system. In this section, we consider using an incomplete factorization as a preconditioner for an iterative method applied to the augmented system (1.3). One possibility is to extend the positive definite limited-memory approach outlined in section 5. This was proposed by Scott and Tůma, who presented a limited-memory signed IC factorization [52].

We discuss the signed IC approach and compare it to the recent LLDL approach of Orban [39].

Scott and Tůma compute an incomplete factorization of the form $\mathrm{LDL}^{\mathrm{T}}$, where L is a lower triangular matrix with positive diagonal entries and D is a diagonal matrix with entries ± 1. In practice, an $\mathrm{LDL}^{\mathrm{T}}$ factorization of

$$\overline{K} = P^T S K S P + \begin{bmatrix} \alpha_1 I & \\ & -\alpha_2 I \end{bmatrix}$$

is computed, where α_1 and α_2 are nonnegative shifts chosen to prevent breakdown of the factorization. The preconditioner is taken to be $\overline{L}D\overline{L}^T$, with $\overline{L} = S^{-1}PL$. Scott and Tůma choose the permutation P not only on the basis of sparsity, but also so that a variable in the $(2,2)$ block of K is not ordered ahead of any of its neighbors in the $(1,1)$ block. The idea here is to try to prevent a small (or zero) pivot candidate from being chosen; see [52] for details of this so-called constrained ordering.

An implementation is available as the HSL package HSL_MI30. As with the IC code HSL_MI35, intermediate memory (R) is optionally used in the construction of the factor L and is then discarded. The user controls the amount of fill allowed in each column of L (*lsize*) and the number of entries in each column of R (*rsize*). The code also includes a range of ordering and scaling options as well as optional dropping parameters (to control the discarding of small entries from L and R).

HSL_MI30 is related to the recent LLDL software developed independently by Orban [39]. The latter extends the ICFS code of Lin and Moré [35] to SQD matrices and thus can be applied to the regularized augmented system (4.2). The main differences between HSL_MI30 and LLDL are the following:

1. LLDL uses a single shift (that is, $\alpha_1 = \alpha_2$).
2. LLDL does not employ intermediate memory (that is, $rsize = 0$).
3. The HSL_MI30 factorization suffers breakdown and is restarted with an increased shift whenever a candidate pivot is not of the expected sign (or is zero). In LLDL there is breakdown only if a pivot is zero; in this case, the shift is increased and the factorization restarted.
4. LLDL does not use a constrained ordering but advises the user to preorder K using a sparsity-preserving ordering, such as approximate minimum degree (AMD) (there is no built-in option for ordering K).
5. LLDL does not include an option to drop small entries during the factorization.
6. HSL_MI35 allows "spare" space from one column of L to be used for the next column (so that if, after dropping, column j of L has $p < lsize$ fill entries, then column $j+1$ may have up to $2 * lsize - p$ fill entries).

For our experiments, we modified HSL_MI30 to implement the same algorithm as LLDL; this facilitates testing using the same ordering and scaling. We refer to this as the LLDL option (although note that if Orban's LLDL package is used, it will return similar but not identical results). Numerical results are given in Tables 6.1 and 6.2. The memory parameters *lsize* and *rsize* were chosen after testing a range of values (with the same *lsize* used for HSL_MI30 and the LLDL option). Recall that $ratio(r)$ is given by (1.4); it enables us to see whether the LSMR stopping criterion is satisfied. We employ l_2-norm scaling of A and AMD ordering; the iterative solver is restarted GMRES (with the restart parameter set to 1000). For tests using the LLDL option, we set $\alpha_1 = \alpha_2 = 1.0$; this choice was made on the basis of experimentation. We set the regularization parameter β to 10^{-6} (recall (4.2)), but our experience is that, for our test set and chosen settings, using a nonzero value has little effect on the quality of the

TABLE 6.1

Results for the signed IC factorization preconditioner HSL_MI30 *run with GMRES to solve the augmented system. lsize and rsize control the memory used by* HSL_MI30*, $nnz(L)$ denotes the number of entries in the factor L, α_2 is the shift for the $(2,2)$ block (in all cases, $\alpha_1 = 0.0$), $time_f$ and $time_t$ are the factorization and total solution times (in seconds), the value of the least-squares objective is $\|r\|_2$, $ratio(r)$ is given by (1.4), and the number of GMRES iterations is itn.*

Problem	*lsize*	*rsize*	$nnz(L)$	α_2	$time_f$	$time_t$	$\|r\|_2$	$ratio(r)$	*itn*
DBIR1	20	0	1.69×10^6	0.004	0.54	0.80	1.667×10^2	3.315×10^{-7}	24
DBIR2	50	0	1.83×10^6	0.008	0.81	1.19	1.665×10^2	2.481×10^{-7}	36
LPL1	20	20	1.05×10^6	0.001	0.59	0.87	7.088×10^1	2.865×10^{-5}	28
NSCT2	100	0	1.80×10^6	0.004	0.65	0.83	1.838×10^2	1.159×10^{-7}	18
PDS-100	20	0	5.21×10^6	0.000	1.65	2.65	2.849×10^2	1.890×10^{-6}	21
PDS-90	20	0	4.87×10^6	0.000	1.56	2.34	2.685×10^2	1.529×10^{-6}	21
beaflw	100	0	9.39×10^4	0.008	0.06	0.24	4.549	1.112×10^{-6}	438
162bit	20	20	1.42×10^5	0.001	0.14	0.28	1.177×10^1	9.558×10^{-6}	161
176bit	20	20	2.95×10^5	0.004	0.35	1.91	1.842×10^1	8.399×10^{-6}	435
192bit	20	50	5.35×10^5	0.000	0.74	1.12	2.485×10^1	9.441×10^{-6}	119
208bit	50	50	1.53×10^6	0.001	3.06	8.42	3.851×10^1	4.564×10^{-6}	460
Maragal_6	20	20	6.44×10^5	0.512	2.25	5.62	1.069×10^1	1.249×10^{-6}	456
Maragal_7	20	20	1.52×10^6	1.024	4.07	7.86	1.369×10^1	1.597×10^{-6}	276
Maragal_8	100	0	3.68×10^6	0.016	1.95	23.1	2.388×10^2	5.237×10^{-6}	676
mri1	20	20	2.25×10^6	0.001	0.95	1.39	2.674×10^1	6.789×10^{-6}	25
mri2	100	0	1.39×10^6	0.512	65.3	99.0	1.413×10^2	1.232×10^{-6}	574
tomographic1	50	0	4.76×10^6	0.001	1.46	11.7	4.192×10^1	1.754×10^{-6}	351

TABLE 6.2

Results for the LLDL *option run with GMRES to solve the augmented system. lsize controls the fill in each column of the factor L, $nnz(L)$ denotes the number of entries in the factor L, $time_f$ and $time_t$ are the factorization and total solution times (in seconds), the value of the least-squares objective is $\|r\|_2$, $ratio(r)$ is given by (1.4), and the number of GMRES iterations is itn. In all cases, $\alpha_1 = \alpha_2 = 1.0$.*

Problem	*lsize*	$nnz(L)$	$time_f$	$time_t$	$\|r\|_2$	$ratio(r)$	*itn*
DBIR1	20	1.27×10^6	0.24	83.9	1.667×10^2	3.774×10^{-7}	3924
DBIR2	20	1.36×10^6	0.26	>600			
LPL1	20	1.04×10^6	0.25	39.7	7.088×10^1	4.087×10^{-5}	1047
NSCT2	100	1.26×10^6	0.32	7.76	1.838×10^2	9.758×10^{-8}	524
PDS-100	20	3.65×10^6	1.36	15.2	2.849×10^2	1.911×10^{-6}	237
PDS-90	0	1.63×10^6	1.13	26.3	2.685×10^2	1.949×10^{-6}	431
beaflw	100	1.05×10^5	0.03	0.47	4.558	6.322×10^{-7}	836
162bit	20	1.15×10^5	0.05	21.3	1.177×10^1	2.530×10^{-6}	5163
176bit	20	2.38×10^5	0.10	149	1.843×10^1	1.101×10^{-6}	20902
192bit	20	4.26×10^5	0.19	250	2.487×10^1	1.981×10^{-6}	24999
208bit	50	1.28×10^6	0.55	>600			
Maragal_6	20	7.44×10^5	0.28	7.67	1.069×10^1	1.184×10^{-6}	730
Maragal_7	20	1.68×10^6	0.62	5.06	1.369×10^1	1.528×10^{-6}	309
Maragal_8	100	2.06×10^6	0.66	>600			
mri1	20	1.94×10^6	0.33	16.1	2.674×10^1	6.999×10^{-6}	506
mri2	100	2.40×10^6	0.37	49.2	1.413×10^2	1.235×10^{-6}	1231
tomographic1	50	2.65×10^6	0.61	>600			

results. We observe that while HSL_MI30 appears robust (it successfully solved all the problems in our test set except BAXTER, which is omitted because we were unable to choose parameters that led to successful convergence), the LLDL option fails to solve a number of problems within our limits of 600 seconds and 100000 iterations.

An alternative to using an IC-based factorization is to employ a general incomplete

indefinite factorization code. Chow and Saad [8] considered the class of incomplete LU preconditioners for solving indefinite problems, and later Li and Saad [33] integrated pivoting procedures with scaling and reordering. Building on this, Greif, He, and Liu [22] recently developed an incomplete factorization package called SYM-ILDL for general sparse symmetric indefinite matrices. Here the system matrix may be any sparse indefinite matrix; no advantage is made of the specific block structure of (1.3). Independently, Scott and Tůma [53] report on the development of incomplete factorization algorithms for symmetric indefinite systems and propose a number of new ideas with the goal of improving the stability, robustness, and efficiency of the resulting preconditioner. Preliminary experiments on our rank-deficient least-squares test problems have found that the indefinite factorization is much less robust than the signed IC approach, and so full results are not included here.

7. Concluding remarks. In this paper, we have used numerical experiments to study solving rank-deficient sparse linear least-squares problems. These are hard problems. In particular, the lack of positive definiteness means that the standard approach of applying a Cholesky solver to the normal equations fails. Our approach is to use existing software to compute the factors of a regularized system and then to employ these factors as a preconditioner with an iterative method to recover the solution of the original problem. We have explored using state-of-the-art parallel sparse direct solvers to compute a complete factorization as well as recent approaches to compute limited-memory incomplete factorizations. Regularization allows a Cholesky-based direct solver to be used to factorize the scaled and shifted normal matrix C, avoiding the need for numerical pivoting that can adversely affect the performance of a sparse indefinite direct solver. However, this requires C to be available. If C cannot be formed or if it is unacceptably dense, the regularized augmented system with the threshold parameter $u = 0.0$ offers a feasible alternative approach (see also [49]).

For very large problems it may not be possible to use a direct solver, and so we have also considered limited-memory IC factorizations. Our results show that IC factorizations of the normal equations computed using the package `HSL_MI35` provide robust preconditioners with significantly sparser factors than those from a complete factorization, but they require a much larger number of LSMR iterations to achieve the requested accuracy. The use of intermediate memory in constructing these IC factors can sometimes significantly enhance the quality of the preconditioner without adding extra fill to the final factors. For the signed IC factorizations of the augmented system, the use of intermediate memory can also be advantageous. In many of our test cases, the total solution time for the signed IC factorizations of the augmented system is greater than for the IC factorizations of the normal equations, but, again, the former has the advantage of avoiding the construction of the normal matrix. We note that our software allows the user to tune a number of parameters, including not only the choice of ordering and scaling but also the amount of memory used by the IC factorization. These choices and the choice of the regularization parameter can significantly affect the quality of the resulting preconditioner, and it may be necessary to experiment with the different options to obtain the best performance for a given application.

Currently, the codes that compute the IC factorizations and then perform the subsequent forward and backward substitutions that are needed when using the factors as preconditioners are serial. As much of the total time is taken by the iterative solver, parallel implementations of the application of the preconditioner are needed. This is currently an area of active research [7].

Finally, although this paper has focused on tackling rank-deficient least-squares problems using sparse direct LL^T and LDL^T solvers and their incomplete factorization counterparts, a number of other approaches are available. In particular, a QR factorization of A may be used, either a complete sparse QR factorization as offered by SuiteSparseQR [9] and qr_mumps [5], or an incomplete QR factorization such as the multilevel incomplete QR (MIQR) factorization of Li and Saad [34]. Moreover, the results reported in [20, 21] suggest that the BA-GMRES approach of Morikuni and Hayami [36, 37] may offer a feasible alternative.

Acknowledgments. I am very grateful to my colleague Nick Gould and to Miroslav Tůma for many useful discussions, and to Michael Saunders for his careful reading of and constructive comments on a draft of this paper. I would also like to thank Dominique Orban for discussions related to using his `LLDL` software for problems with A rank-deficient. Many thanks also to two anonymous reviewers for their helpful feedback that led to improvements in both the text and the numerical results.

REFERENCES

[1] M. Arioli, I. S. Duff, and P. P. M. Rijk, *On the augmented system approach to sparse least-squares problem*, Numer. Math., 55 (1989), pp. 667–684, https://doi.org/10.1007/BF01389335.

[2] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477, https://doi.org/10.1006/jcph.2002.7176.

[3] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996, https://doi.org/10.1137/1.9781611971484.

[4] R. Bru, J. Marín, J. Mas, and M. Tůma, *Preconditioned iterative methods for solving linear least squares problems*, SIAM J. Sci. Comput., 36 (2014), pp. A2002–A2022, https://doi.org/10.1137/130931588.

[5] A. Buttari, *Fine-grained multithreading for the multifrontal QR factorization of sparse matrices*, SIAM J. Sci. Comput., 35 (2013), pp. C323–C345, https://doi.org/10.1137/110846427.

[6] Y. Chen, T. A. Davis, W. H. Hager, and S. Rajamanickam, *Algorithm* 887: *CHOLMOD, supernodal sparse Cholesky factorization and update/downdate*, ACM Trans. Math. Software, 35 (2008), pp. 22:1–22:14, https://doi.org/10.1145/1391989.1391995.

[7] E. Chow and A. Patel, *Fine-grained parallel incomplete LU factorization*, SIAM J. Sci. Comput., 37 (2015), pp. C169–C193, https://doi.org/10.1137/140968896.

[8] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414, https://doi.org/10.1016/S0377-0427(97)00171-4.

[9] T. A. Davis, *Algorithm* 915: *SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization*, ACM Trans. Math. Software, 38 (2011), pp. 8:1–8:22, https://doi.org/10.1145/2049662.2049670.

[10] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25, https://doi.org/10.1145/2049662.2049663.

[11] I. S. Duff, *MA*57*—A code for the solution of sparse symmetric definite and indefinite systems*, ACM Trans. Math. Software, 30 (2004), pp. 118–154, https://doi.org/10.1145/992200.992202.

[12] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner, *The factorization of sparse symmetric indefinite matrices*, IMA J. Numer. Anal., 11 (1991), pp. 181–204, https://doi.org/10.1093/imanum/11.2.181.

[13] I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 889–901, https://doi.org/10.1137/S0895479897317661.

[14] D. C.-L. Fong and M. Saunders, *LSMR: An iterative algorithm for sparse least-squares problems*, SIAM J. Sci. Comput., 33 (2011), pp. 2950–2971, https://doi.org/10.1137/10079687X.

[15] A. George and M. A. Saunders, *Solution of Sparse Linear Equations Using Cholesky Factors of Augmented Systems*, Research Report SOL 99-1, Department of EESOR, Stanford University, Stanford, CA, 1999.

[16] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders, *Preconditioners for indefinite systems arising in optimization*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 292–311, https://doi.org/10.1137/0613022.

[17] P. E. Gill, M. A. Saunders, and J. R. Shinnerl, *On the stability of Cholesky factorization for symmetric quasidefinite systems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 35–46, https://doi.org/10.1137/S0895479893252623.

[18] G. H. Golub and C. F. Van Loan, *Unsymmetric positive definite linear systems*, Linear Algebra Appl., 28 (1979), pp. 85–97, https://doi.org/10.1016/0024-3795(79)90122-8.

[19] N. I. M. Gould, D. Orban, and Ph. L. Toint, *CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization*, Comput. Optim. Appl., 60 (2015), pp. 545–557, https://doi.org/10.1007/s10589-014-9687-3.

[20] N. I. M. Gould and J. A. Scott, *The State-of-the-Art of Preconditioners for Sparse Linear Least Squares Problems: The Complete Results*, Technical Report RAL-TR-2015-009, Rutherford Appleton Laboratory, Oxfordshire, UK, 2015; available online at http://purl.org/net/epubs/work/23411221.

[21] N. I. M. Gould and J. A. Scott, *The state-of-the-art of preconditioners for sparse linear least-squares problems*, ACM Trans. Math. Software, 43 (2017), pp. 36:1–36:35, https://doi.org/10.1145/3014057.

[22] C. Greif, S. He, and P. Liu, *SYM-ILDL: Incomplete LDL^T Factorization of Symmetric Indefinite and Skew-Symmetric Matrices*, Technical report, Department of Computer Science, The University of British Columbia, Vancouver, Canada, 2015; software available from https://www.cs.ubc.ca/~inutard/html/.

[23] A. Gupta, *WSMP Watson Sparse Matrix Package (Part* II: *Direct Solution of General Sparse Systems)*, Technical Report RC 21888 (98472), IBM T. J. Watson Research Center, Yorktown Heights, NY, 2000.

[24] J. D. Hogg, E. Ovtchinnikov, and J. A. Scott, *A sparse symmetric indefinite direct solver for GPU architectures*, ACM Trans. Math. Software, 42 (2016), pp. 1:1–1:25, https://doi.org/10.1145/2756548.

[25] J. D. Hogg, J. K. Reid, and J. A. Scott, *Design of a multicore sparse Cholesky factorization using DAGs*, SIAM J. Sci. Comput., 32 (2010), pp. 3627–3649, https://doi.org/10.1137/090757216.

[26] J. D. Hogg and J. A. Scott, *The Effects of Scalings on the Performance of a Sparse Symmetric Indefinite Solver*, Technical Report RAL-TR-2008-007, Rutherford Appleton Laboratory, Oxfordshire, UK, 2008.

[27] J. D. Hogg and J. A. Scott, *HSL_MA97: A Bit-Compatible Multifrontal Code for Sparse Symmetric Systems*, Technical Report RAL-TR-2011-024, Rutherford Appleton Laboratory, Oxfordshire, UK, 2011; available online at http://purl.org/net/epubs/work/61445.

[28] J. D. Hogg and J. A. Scott, *A Study of Pivoting Strategies for Tough Sparse Indefinite Systems*, Technical Report RAL-TR-2012-009, Rutherford Appleton Laboratory, Oxfordshire, UK, 2012; available online at http://purl.org/net/epubs/work/63152.

[29] J. D. Hogg and J. A. Scott, *New parallel sparse direct solvers for multicore architectures*, Algorithms, 6 (2013), pp. 702–725, https://doi.org/10.3390/a6040702.

[30] J. D. Hogg and J. A. Scott, *Pivoting strategies for tough sparse indefinite systems*, ACM Trans. Math. Software, 40 (2013), pp. 4:1–4:19, https://doi.org/10.1145/2513109.2513113.

[31] *The HSL Mathematical Software Library*, a collection of Fortran codes for large-scale scientific computation, http://www.hsl.rl.ac.uk (2016).

[32] G. Karypis and V. Kumar, *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices (Version* 4.0*)*, Technical report, Department of Computer Science and Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.

[33] N. Li and Y. Saad, *Crout versions of ILU factorization with pivoting for sparse symmetric matrices*, Electron. Trans. Numer. Anal., 20 (2005), pp. 75–85.

[34] N. Li and Y. Saad, *MIQR: A multilevel incomplete QR preconditioner for large sparse least-squares problems*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 524–550, https://doi.org/10.1137/050633032.

[35] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45, https://doi.org/10.1137/S1064827597327334.

[36] K. Morikuni and K. Hayami, *Inner-iteration Krylov subspace methods for least squares problems*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1–22, https://doi.org/10.1137/110828472.

[37] K. Morikuni and K. Hayami, *Convergence of inner-iteration GMRES methods for rank-deficient least squares problems*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 225–250, https://doi.org/10.1137/130946009.

[38] *MUMPS* 5.0.0*: A Multifrontal Massively Parallel Sparse Direct Solver*, http://mumps-solver.org (2015).
[39] D. Orban, *Limited-memory LDL^T factorization of symmetric quasi-definite matrices with application to constrained optimization*, Numer. Algorithms, 70 (2015), pp. 9–41, https://doi.org/10.1007/s11075-014-9933-x.
[40] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629, https://doi.org/10.1137/0712047.
[41] C. C. Paige and M. A. Saunders, *Algorithm* 583: *LSQR: Sparse linear equations and least squares problems*, ACM Trans. Math. Software, 8 (1982), pp. 195–209, https://doi.org/10.1145/355993.356000.
[42] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71, https://doi.org/10.1145/355984.355989.
[43] *PARDISO* 5.0.0 *Solver Project*, http://www.pardiso-project.org (2014).
[44] J. K. Reid and J. A. Scott, *An out-of-core sparse Cholesky solver*, ACM Trans. Math. Software, 36 (2009), pp. 9:1–9:33, https://doi.org/10.1145/1499096.1499098.
[45] D. Ruiz, *A Scaling Algorithm to Equilibrate both Rows and Columns Norms in Matrices*, Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, Oxfordshire, UK, 2001.
[46] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003, https://doi.org/10.1137/1.9780898718003.
[47] Y. Saad and M. H. Schulz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869, https://doi.org/10.1137/0907058.
[48] M. A. Saunders, *Solution of sparse rectangular systems using LSQR and CRAIG*, BIT, 35 (1995), pp. 588–604, https://doi.org/10.1007/BF01739829.
[49] M. A. Saunders, *Cholesky-based methods for sparse least squares: The benefits of regularization*, in Linear and Nonlinear Conjugate Gradient-Related Methods, Proceedings of the AMS-IMS-SIAM Summer Research Conference, L. Adams and J. L. Nazareth, eds., SIAM, Philadelphia, 1996, pp. 92–100.
[50] J. Scott and M. Tůma, `HSL_MI28`: *An efficient and robust limited-memory incomplete Cholesky factorization code*, ACM Trans. Math. Software, 40 (2014), pp. 24:1–24:19, https://doi.org/10.1145/2617555.
[51] J. Scott and M. Tůma, *On positive semidefinite modification schemes for incomplete Cholesky factorization*, SIAM J. Sci. Comput., 36 (2014), pp. A609–A633, https://doi.org/10.1137/130917582.
[52] J. Scott and M. Tůma, *On signed incomplete Cholesky factorization preconditioners for saddle-point systems*, SIAM J. Sci. Comput., 36 (2014), pp. A2984–A3010, https://doi.org/10.1137/140956671.
[53] J. Scott and M. Tůma, *Improving the stability and robustness of incomplete symmetric indefinite factorization preconditioners*, Numer. Linear Algebra Appl., (2017), https://doi.org/10.1002/nla.2099.
[54] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, Internat. J. Numer. Methods Engrg., 23 (1986), pp. 239–251, https://doi.org/10.1002/nme.1620230208.
[55] A. van der Sluis, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1969), pp. 14–23, https://doi.org/10.1007/BF02165096.
[56] R. J. Vanderbei, *Symmetric quasidefinite matrices*, SIAM J. Optim., 5 (1995), pp. 100–113, https://doi.org/10.1137/0805005.